4.1 INTRODUCTION

The relational model was introduced by EF Codd in 1970. A relational database consist of a collection of tables called relation. A relation consist of following:

1. **Tuple** – Row of a relation
2. **Attribute** – Column of a relation
3. **Domain** – Set of allowed value for each attributes
4. **Degree** – Number of columns in a relation
5. **Cardinality** – Number of rows in a relation

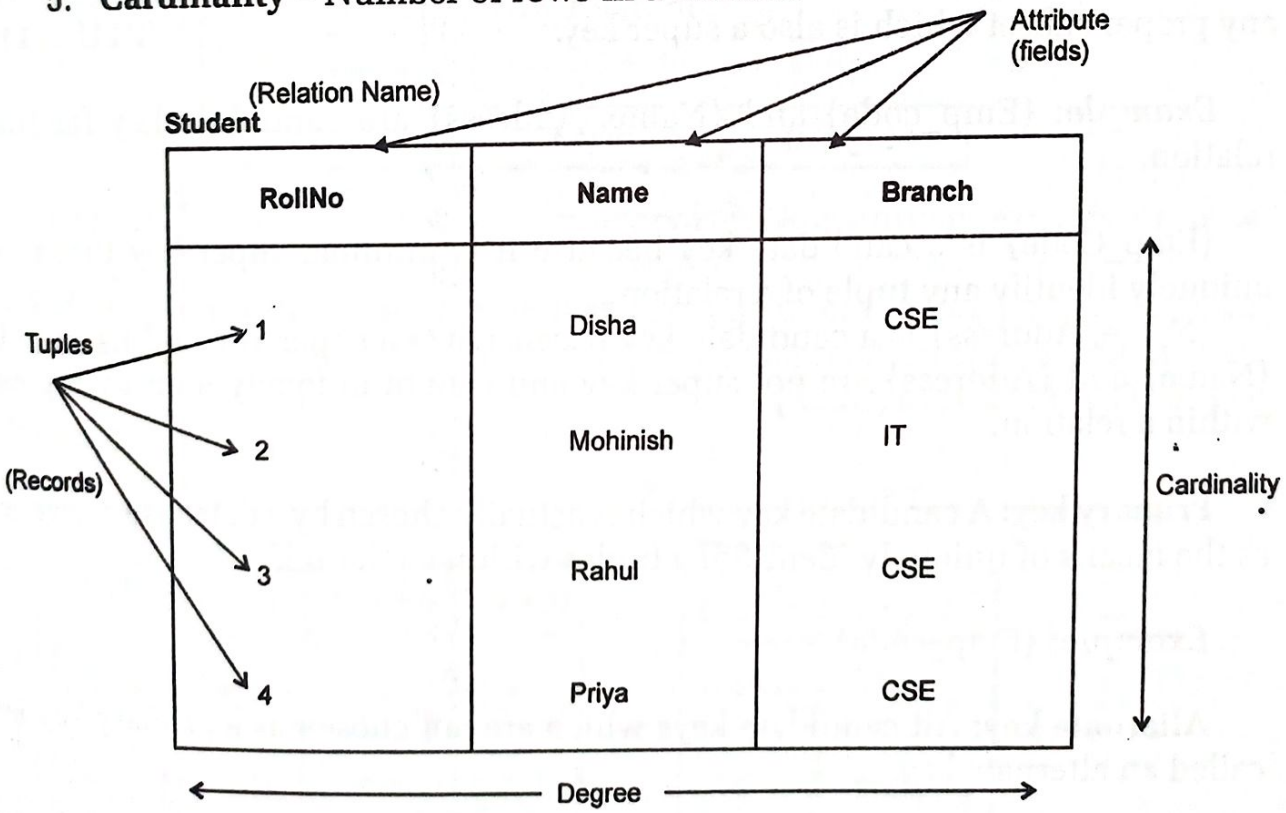


Fig. 4.1: Structure of a Relation

4.2 KEYS

(GGSIPU, 2013)

A key is an attribute or a set of attributes which can uniquely identify each tuple of a relation.

Super key: It is an attribute or a set of attribute whose combined value uniquely identifies a tuple within a relation.

Example: Consider the following relation

Employee

Emp_Code	Name	Address	Dept_Id

For the above relation following super key can exist:

(a) {Emp_code, Name, Address, Dept_Id}
(b) {Emp_code, Name, Address}
(c) {Name, Address}
(d) {Emp_code} etc.

Each of the above combination of attribute can uniquely identify any tuple in a relation. (We assume no two employee with same name and address exist)

Candidate key: It is a minimal super key i.e. a super key which does not have any proper subset which is also a super key. (PTU 2011)

Example: {Emp_code} and {Name, Address} are candidate key for the relation.

{Emp_Code} is a candidate key because it is minimal super key that can uniquely identify any tuple of a relation.

{Name, Address} is a candidate key because it is a super key and its subset {Name} and {Address} are not super key and cannot uniquely identify tuples within a relation.

Primary key: A candidate key which is actually chosen by a database designer as the means of uniquely identifying tuples within a relation.

Example: {Emp_code}

Alternate key: All candidate keys which are not chosen as a primary key is called an alternate key.

Example: {Name, Address}

Composite primary key: A primary key formed by combination of one or more attributes.

Example: {Name, Address}

Foreign key: When a primary key of one relation exist as an attribute in another relation then, this attribute is called a foreign key. Foreign key is used to establish a relationship between two tables.

***Example*:** Consider one more relation

Department

Dept_Id	Dept_Name

Dept_Id which is a primary key of the relation department exist as an attribute in the relation employee. So, Dept_Id is a foreign key in employee relation.

4.3 INTEGRITY CONSTRAINTS

(GGSIPU 2008, 2011; MDU Dec., 2009)

These constraints are applied to all instances of the database to keep the data consistent.

Entity Integrity Constraint: It states that the value of attribute of a primary key cannot be null. **(MDU, May 2011)**

***Example*:** Consider the following relation

Employee

E_Id	Name	Dept_Id

in which E_Id is designated as the primary key. Now, any value in this attribute cannot be null.

Referential Integrity Constraint: It states that either the foreign key can have only those values that match the primary key of other relation or it has null value. **(MDU, May 2011; PTU 2011)**

Example:

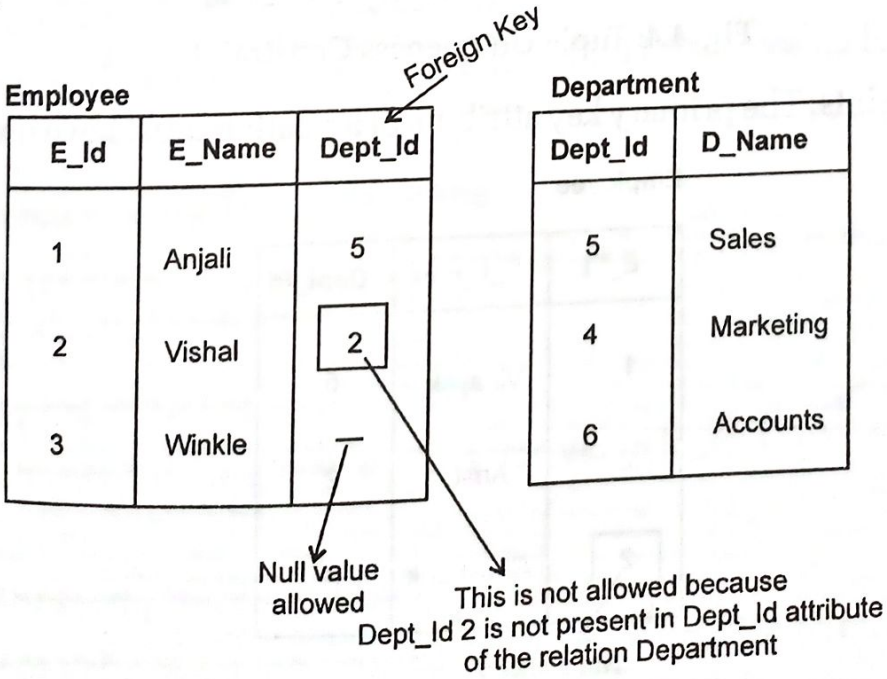


Fig. 4.2: Referential Integrity Constraint

Domain Constraint: It states that the value of each attribute must be an atomic value from the domain of that attribute.

***Example*:** If Name is defined to be character, then it cannot take integer value

Employee

E_Id	Name	Dept_Id
1	Maya	7
2	5	3

Integer value not allowed

Fig. 4.3

Tuple Uniqueness Constraint: All tuples in a relation must be distinct.

Example:

Employee

E_Id	E_Name	Dept_Id
1	Vinayak	5
2	Amit	2
2	Amit	2

Duplicate tuple not allowed

Fig. 4.4: Tuple Uniqueness Constraint

Key Constraints: The primary key attribute of a relation must have unique values.

Employee

E_Id	E_Name	Dept_Id
1	Vinayak	5
2	Amit	2
2	Shashank	4

Not allowed

Fig. 4.5: Key Constraint

4.4 CODD RULES

Dr. E.F. Codd proposed a set of rules that are necessary for design of all relational database system.

Rule 1: The information rule

All data in database should be present in form of a table.

Rule 2: Guaranteed access rule

All data can be accessed from a database without any ambiguity. Any data item is accessed using a combination of table name, column name and primary key values.

Rule 3: Systematic treatment of null values

Null values can be used to represent missing information. Null value must be distinct from zero and should be independent of data type.

Rule 4: Active online relational catalog

The database represented in form of table can be queried using the query language.

Rule 5: Comprehensive data sub-language role

At least one language should be supported by RDBMS that include functionality of data definition, manipulation, integrity rules, authorisation and transaction.

Rule 6: View updating rule

All view that are updateble can be updated through the system.

Rule 7: High level insert, update and delete

All insertion, update and delete operation should be supported for any set rather than for a single row.

Rule 8: Physical data independence

Application programme, operation or data is logically unaffected when physical access method are changed.

Rule 9: Logical data independence

How data is viewed should remain the same when logical structure of a database is changed.

Rule 10: Integrity independence

The language defined by the database should be capable of enforcing integrity rules.

Rule 11: Distribution Independence

Application programme and user operation should be unaffected by the location of data and any changes made to distribution of the data.

Rule 12: Non sub-version

Integrity rule must not be bypassed by using lower level language.

4.5 DIFFERENCE BETWEEN DBMS AND RDBMS

Concept	DBMS	RDBMS
Relation between tables	Maintained Programmatically	Relation between table is maintained by common attribute
Multi User	generally do not support multiple user simultaneously at same time	Multiple user concept supported
Data Security	Generally not supported	Multiple level of security
Abstract View	Generally not supported	Abstract view supported
Database Storage	A database is stored in a single file	A database can spread across multiple tables
Limit of Data Storage	Cannot store very large amount of data	Can store very large amount of data
Access to database file	User can have access to stored file	User do not have access to low level files
Distributed database	Not supported	Supported
Codd's Rule	Do not satisfy all Codd rule	Support all Codd rule
System Requirements	Can run with simple system configuration	Require powerful hardware
Example	Sybase, Foxpro	Oracle, MS SQL Server DB2

4.6 RELATIONAL ALGEBRA

(GGSIPU, 2010; MDU, May 2012; KU Dec 2005, UPTU 2011-12; PTU 2011, 2009)

Relational Algebra is a formal language for relational model. It is a procedural query language in which the user instructs the system to perform some sequence of operations on database to compute the desired result.

Relational algebra is a set of operations on relational databases that allow retrieval of data. It uses a set of relational operators which takes as input one or more relations perform some operations and returns the required result which is also a relation. Relational algebra serves as the basis of SQL and query optimization.

To explain the various operations of relational algebra we would use following banking schema.

bank (b_Id, b_city)
customer (c_name, c_city, c_state)
account (acc_no, b_Id, balance)
deposit (c_name, acc_no)
borrow (c_name, l_no)
loan (l_no, b_Id, amount)

bank

b_Id	b_city
206	Panipat
207	Gurgaon
208	Noida
208	Faridabad
210	Noida

customer

c_name	c_city	c_state
Ankit	Panipat	Haryana
Rajesh	Noida	U.P.
Dheeraj	Gurgaon	Haryana
Ravi	Noida	U.P.
Vinod	Faridabad	Haryana

account

acc_no	b-Id	balance
A-95	206	20,000
A-56	207	50,000
A-32	205	7,000
A-42	208	80,000
A-80	208	65,000

deposit

c_name	acc_no
Ankit	A-95
Rajesh	A-32
Dheeraj	A-56
Ravi	A-42
Vinod	A-80

borrow

c_name	l_no
Ankit	L-25
Ravi	L-33

loan

l_no	b_Id	amount
L-25	206	50,000
L-33	208	70,000

Fig. 4.6: Banking Schema Relations

4.6.1 Selection Operation

Selection operation is a unary operation that select rows (tuples) that satisfy some given condition. It is denoted by σ (sigma)

***Example* 1:** Generate list of customers who live in Noida

The query is

$\sigma_{c\_city = \text{'Noida'}}$ **(customer)**

In the above query

(i) σ indicates the select operation
(ii) c_city = 'Noida' is the condition to be satisfied.
(iii) customer specified in circular braces indicates the relation where we could find the customer information on which condition is to be applied.

Result of the query will be

c_name	c_city	c_state
Rajesh	Noida	U.P.
Ravi	Noida	U.P.

Fig. 4.7: Result of Selection Operation

Example 2: Generate account information where balance is less than Rs. 10,000 or greater than Rs. 60,000.

Query is

$$\sigma_{\text{balance} < 10000 \text{ V balance} > 60,000} \text{ (account)}$$

Result of the query will be

acc-no	b_Id	balance
A-32	208	70,000
A-42	208	80,000
A-42	209	65,000

Fig. 4.8: Result of Conditional Selection Operation

4.6.2 Projection Operation

Project operation is a unary operation that displays only some specified attribute of a relation. It is denoted by π (pie).

Example 3: Generate list of account number and balance in each account

Query is

$$\pi_{\text{acc\_no, balance}} \text{ (account)}$$

In above query

(i) π indicates project operation
(ii) acc_no, balance are attribute names to be displayed.
(iii) account is a relation name where the above attributes are present.

Result of the query will be

acc_no	balance
A-95	20,000
A-56	50,000
A-38	7,000
A-42	80,000
A-80	65,000

Fig. 4.9: Result of Projection Operation.

***Example* 4:** Generate loan number of customer who have loan more than Rs. 50,000

Query is

$\pi_{l\_no}(\sigma_{amount > 50,000}(\textbf{loan}))$

Result of the query will be

loan
L-33

Fig. 4.10: Result of Selection and Projection Operation

4.6.3 Union Operation

Let n be the arity (number of column) of relation A and B. Union of A and B is the set of tuples with arity n, which are either in A or in B or in both A and B. It is denoted by ∪.

***Example* 5:** Find names of customers who have a loan, an account or both from the bank.

Query is

$\pi_{c\_name}(\textbf{borrow}) \cup \pi_{c\_name}(\textbf{deposit})$

Result of the query will be

c_name
Ankit
Ravi
Rajesh
Dheeraj
Vinod

Fig. 4.11: Result of Union Operation

Here, borrow relation is used because name of customer who have taken loan exist in borrow relation. Similarly deposit relation is used because names of the customer who have an account in bank exist in deposit relation.

In the result first all the names which exist in borrow relation are displayed and then the names of customer who have an account and do not have any loan are displayed.

4.6.4 Set Intersection Operation

Set intersection of two relation A and B are set of all types which are in both A and B.

Example 6: Find name of all the customer who have both an account and a loan in the bank.

Query is

$$\pi_{c\_name}(deposit) \cap \pi_{c\_name}(borrow)$$

Result of the query will be

c_name
Ankit
Ravi

Fig. 4.12: Result of Set Intersection Opeation

4.6.5 Set Difference Operation

Consider two relations A and B. Set difference of relation A and B are set of tuples which are in A but not in B. It is denoted by '–'.

Here, arity of A and B should be same.

Example 7: Find name of customer who have an account but do not have any loan.

Query is

$$\pi_{c\_name}(deposit) - \pi_{c\_name}(borrow)$$

Result of the query will be

c_name
Rajesh
Dheeraj
Vinod

Fig. 4.13: Result of Set Difference Opeation

4.6.6 Cartesian Product

It takes each tuple from the first relation and concatenates it with every tuple of the second relation. It is denoted by (X).

Consider the following relations

A

a	b	c
a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}

B

d	e
b_{11}	b_{12}
b_{21}	b_{22}

Cartesian product of A and B is

a	b	c	d	e
a_{11}	a_{12}	a_{13}	b_{11}	b_{12}
a_{11}	a_{12}	a_{13}	b_{21}	b_{22}
a_{21}	a_{22}	a_{23}	b_{11}	b_{12}
a_{21}	a_{22}	a_{23}	b_{21}	b_{22}

A × B = 2 × 2 Tuples

Fig. 4.14: Example of Cartesian Product

***Example* 8:** Find name of all customer who have loan at branch with branch Id 206

Query is

$$\pi_{c\_name}(\sigma_{b\_ID = 206}(\sigma_{borrow.l\_no = loan.l\_no}(\textbf{borrow} \times \textbf{loan})$$

or

$$\pi_{c\_name}(\sigma_{borrow.l\_no = loan.l\_no}[\sigma_{b\_ID} = 206\ (\textbf{borrow} \times \textbf{loan})]$$

Explanation

Result of the query **(borrow × loan)** will be

c_name	borrow.l_no	loan.l_no	b_Id	amount
Ankit	L-25	L-25	206	50,000
Ankit	L-25	L-33	208	70,000
Ravi	L-33	L-25	206	50,000
Ravi	L-33	L-33	208	70,000

Result of $\sigma_{b\_Id = 206}$ **(borrow × loan)** will be

c_name	borrow.l_no	loan.l_no	b_Id	amount
Ankit	L-25	L-25	206	50,000
Ravi	L-33	L-25	206	50,000

Result of $\sigma_{borrow.l\_no = loan.l\_no}[\sigma_{b\_Id = 206}(borrow \times loan)]$ will be

c_name	borrow.l_no	loan.l_no	b_Id	amount
Ankit	L-25	L-25	206	50,000

Final Result $\pi_{c\_name}(\sigma_{borrow.l\_no = loan.l\_no}[\sigma_{b\_Id = 206}(borrow \times loan)]$

c_name
Ankit

Fig. 4.15: Result of Cartesian Product

Customer name who have taken loan exist in borrow relation whereas branch Id (b_Id) from which they have taken loan exist in loan relation. So, we require information from both these relations. For this a Cartesian product operation is performed on relations borrow and loan. The new relationship formed contains all information from both the relations where b_Id = 206. Now in the answer final result corresponds to those tuples where borrow.l_no and loan.l_no match in (borrow × loan) relation.

Ravi is not a answer because he do not have loan from b_Id = 206 but from b_Id = 208.

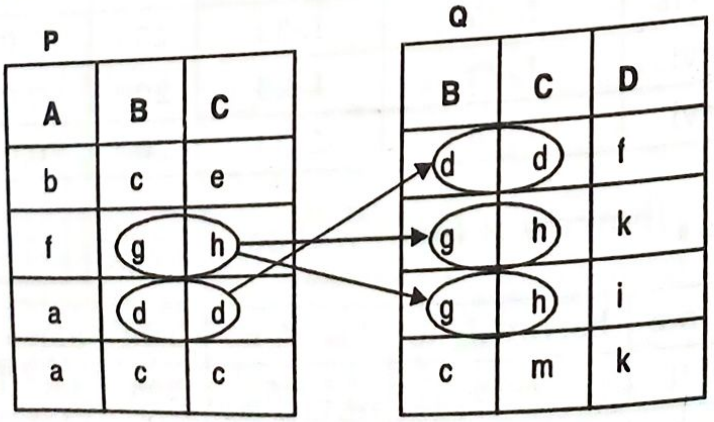
***Example* 9:** Find name of customer who have a loan at branch_Id = 208 but do not have an account in any branch of bank.

$$\pi_{c\_name}(\sigma_{b\_Id = 208}[\sigma_{borrow.l\_no = loan.l\_no}(borrow \times loan) - \pi_{c\_name}(deposit)]$$

4.6.7 Natural Join

Natural join operation is same as the Cartesian product operation i.e. it joins two relation but the difference is that it automatically forces equality on those attributes that appears in both the relations and then also removes duplicate attributes. It is denoted by (⋈). If join attribute of two relations have same name then they need not be specified.

Consider relations



$P \bowtie Q$

A	B	C	D
f	g	h	k
f	g	h	i
a	d	d	f

Fig. 4.16: Example of Natural Join Operation

***Example* 10:** Find the name of customer who have a loan at branch with branch Id 206.

Result

$$\pi_{c\_name} [\sigma_{b\_Id = 206} (borrow \bowtie loan)]$$

***Example* 11:** Find name of customer who have an account at both branch Id 206 and branch Id 208.

Result

$$\pi_{c\_name} [\sigma_{b\_Id = 206} (deposit \bowtie account)] \cap \pi_{c\_name} (\pi_{b\_Id = 208} (deposit \bowtie account)]$$

4.6.8 Division Operation (GGSIPU 2013, 2012)

It divides a relation of degree (m+n) by a relation B of degree n and produces a relation of degree m. This operation is useful in queries that includes "for all" phrase. It is denoted by ÷.

If relation R1 defined over attribute set X and relation R2 is defined over attribute set Y such that $Y \subseteq X$.

The division operation produces a relation R(Z) that include all tuples Z in R1 (X) that appear in R1 in combination with every tuple from R2(X), where $Z = X \cup Y$.

Teacher	Subject
Geeta	Data Structure
Nishant	Microprocessor
Geeta	Networking
Manoj	Database

Subject
Data Structure
Networking

To find name of teacher who specialise in both data structure and networking we perform a divide operation.

Teacher
Geeta

Fig. 4.17: Example of Divsion Operation

***Example* 12:** Find name of all customers who have an account at both branches located in Noida.

$$\pi_{c\_name,\ b\_Id}\ (\text{deposit} \bowtie \text{account}) \div \pi_{b\_Id}\ (\sigma_{b\_city\ =\ 'Noida'}\ (\text{bank}))]$$

c_name	b_Id
Ankit	206
Rajesh	208
Dheeraj	207
Ravi	208
Vinod	209

÷

b_Id
208
210

Result of the query will be

c_name

Fig. 4.18: Result of Division Operation

There is no customer who have account in both the branches in Noida.

4.6.9 Rename

Rename operation is used to give names to relational algebra expressions 'E'. It is denoted by ρ_x (E)

Example 13: Find the largest balance

$$\pi_{balance}(\text{account}) - \pi_{account.balance}\ (\sigma_{account.balance\ <\ x.balance}\ (\text{account} \times \rho_x\ (\text{account})$$

Result of the query will be

balance
80000

Fig. 4.19: Result of Rename Operation

In the above query we need to compare each account balance with all other account balances and select the lower balance of the two balances. For this, we need to perform a Cartesian product account x account (we rename the second

4.6.11 Outer Join Operation (GGSIPU 2013; UPTU 2011-12)

Outer join is the same as natural join operations but it also deals with the missing information caused by natural join operation. In natural join the tuples of participating relations are lost in result whenever a matching tuple is not found in both participating retations but in outer join if there is a mismatch, the tuple is retained and the missing value is represented by a NULL. Consider the following relation:

Teacher

T_Id	Name	Dept_Id
12	Rajesh	4
26	Mukesh	2
56	Pankaj	7

Department

Dept_Id	DeptName
4	CSE
2	IT
5	ECE

Fig. 4.21: Teacher and Department Relation

***Join* (Same as Natural Join)**
Teacher ⋈ Department

T_Id	Name	Dept_Id	Dept Name
12	Rajesh	4	CSE
26	Mukesh	2	IT

Fig. 4.22: Result of Join Operation

Left Outer Join

It takes all tuples of relation on left side and that, which did not match with any tuple on right relation. Left Outer Join is denoted by ⟕.

Teacher ⟕ Department

T_Id	Name	Dept_Id	Dept Name
12	Rajesh	4	CSE
26	Mukesh	2	IT
56	Pankaj	7	NULL

Fig. 4.23: Result of Left Outer Join

Right Outer Join

It takes all tuples of relation that are on right side and that which did not match with any tuple on left side. Right outer join is denoted by ⟖

Teacher ⟖ Department

T_Id	Name	Dept_Id	Dept Name
12	Rajesh	4	CSE
26	Mukesh	2	IT
NULL	NULL	5	ECE

Fig. 4.24: Result of Right Outer Join

Full Outer Join

It takes all tuples from left side relation and right side relation whether they match with each other or did not match. Full Outer Join is denoted by ⟗

Teacher ⟗ Department

T_Id	Name	Dept_Id	DeptName
12	Rajesh	4	CSE
26	Mukesh	2	IT
56	Pankaj	7	NULL
NULL	NULL	5	ECE

Fig. 4.25: Result of Full Outer Join

4.6.12 Generalized Projection

Generalized projection operation extends the projection operation by allowing arithmetic operation to be used in the projection list.

Example: Suppose we have a relation product having attributes ProductID, ProductName, Price and Discount. Now to find ProductId, Product_Name and discounted_Price, we can write following expression.

$$\pi_{\text{ProductId, ProductName, (Price – Discount) as discounted\_price}} \textbf{(Product)}$$

4.7 RELATIONAL CALCULUS

Relation calculus is a formal language for relation model. It is a non procedural query language that provide a higher level declarative notation for specifying relational queries. In relational calculus, a query is expressed as a formula consisting of a number of variables and an expression involving this expression. The formula describes properties of result relation to be obtained. It is upto RDBMS to decide how to evaluate this formula by converting non procedural queries to equivalent procedural queries.

4.7.1 Tuple Relational Calculus

(GGSIPU, 2010, 2011)

Tuple relational calculus is a non-procedural query language in which we specify what is to be retrieved rather than how to retrieve it. This query can be easily used by a non-expert person. The tuple relational calculus is based on specifying number of tuple variable. Each such tuple variable normally ranges over a particular database relation i.e. a variable my take any individual tuple from that relation as its value. Each query in tuple calculus is of the form.

$\{t \mid \text{condition (t)}\}$

where *t* is a tuple variable, condition (t) is a conditional expression involving t and result of this query is set of all tuples t that satisfy condition (t).

General expression for tuple calculus is

$$\{t_1, A_j, t_2, A_k, \ldots..t_n.\ A_m \mid \text{condition } (t_1, t_2, \ldots..t_n, t_{n+1}, \ldots.t_{n+m})\}$$

Where $t_1, t_2, \ldots..t_n, t_{n+1}, \ldots.t_{n+m}$ are tuple variable, each A_i is an attribute of the relation on which t_i ranges and condition is a condition or formula. A formula is made up of atom which are:

(i) An atom of form R(t) where R is a relation and t is tuple value. It means t is a tuple in relation of R.
(ii) An atom of form $t_i.A \otimes t_j.B$ where $\otimes$ is any one comparison operator from the set
$\{<, \leq, >, \geq, =, \neq\}$.
(iii) An atom of form t_i. A $\otimes$ constant or constant $\otimes$ t_i.A.

Each of above atoms evaluates to either TRUE or FALSE for a specific combination of tuples and called truth values of an atom.

Now, a formula (also known as **well-formed formula (WFF)** is made up of one or more atom where **(KU, Dec 2005)**

(i) Every atom is a WFF.
(ii) If F_1 and F_2 are WFF then so are (F_1 and F_2), (F_1 or F_2), Not (F_1) and Not (F_2).
(iii) If F(n) is a WFF and n is free, then $\exists x$ (F(x)) and $\forall$ x(F(n)) are also WFF.

Quantifiers and Tuple Variable

Existential quantifier ($\exists$) – It means 'there exist'. It is used in formula that must be true for at least one instance.

Universal quantifier ($\forall$) – It means 'for all'. Every tuple in universe of tuple must make formula True to make quanlified formula True.

Free variable – A variable in tuple calculus is said to be a free variable when

it appears left to vertical bar i.e. it is not quantified by the existential (∃) or the universal quantifier (∀).

Bound variable – A variable in tuple calculus is said to be bound variable if it appears on right side of vertical bar. It is quantified by the existential (∃) or the universal (∀) quantifier.

Consider the following relations

Student (Roll, Name, Age, Address, Course, Year, Gender, Dept No.)
Department (DeptNo, DName)

Example 1. Find out the name and roll no of BTECH students of first year.

{t. Roll, t.Name | Student (t) AND t.course = 'BTECH' AND t.year = 1}

attribute required in result

Range relation which is true whenever value of t is a tuple in Student relation

Example 2. Retrieve student information who belong to Department number 5.

{t | Student(t) AND t.DeptNo=5}

To display all attributes

Example 3. Retrieve the roll no, name of all girl students in CSE department

{s.Roll, s.Name | Student(s) AND s.Gender = 'F'
AND (∃d) (Department(d) AND d.DName = 'CSE'
AND d.DeptNo=s.DeptNo}

In the above query 's' is a free variable as it is not quantified and appears on left side of vertical bar, whereas d is a bound variable.

Example 4. Determine the departments that do not have any girl student less than 18 years of age.

{d.DName | Department(d) AND NOT (∃s)
(Student(s) AND s.Gender = 'F' AND
s.Age < 18 AND s.DeptNo=d.DeptNo)}

4.7.2 Domain Relational Calculus (GGSIPU 2010, 2011)

Domain relational calculus is also non-procedural query language in which variable range over single values from domain of attributes i.e. domain variables that takes on values from an attribute's domain rather than values for an entire tuple. It is used in QBE. A domain calculus expression is:

$$\{x_1, x_2, \ldots x_n \mid \text{Condition } (x_1, x_2, \ldots x_n, x_{n+1}, \ldots x_{n+m}\}$$

Where $x_1, x_2, \ldots x_n, x_{n+1} \ldots x_{n+m}$ are domain variables that range over domains of attributes and condition is a condition or formula of domain relational calculus.

A formula is made up of atoms and can have any one of the following form:

(i) $R(x_1, x_2, \ldots x_n)$ where R has degree n and each x_i, $i \le i \le n$ is a domain variable. It means $x_1, x_2, \ldots x_n$ are tuples in relation R and x_i is value of i^{th} attribute.

(ii) $X \otimes Y$ where $\otimes$ is any one comparison operator from the set. $\{=, \neq, <, \geq, \leq, \geq\}$ and x and y are domain variables.

(iii) $X \otimes$ constant or constant $\otimes x$

A well formed formula is defined as

(a) Every atom is a WFF

(b) If F_1 and $_2$ are formula so are (F_1 AND F_2), (F_1 OR F_2), NOT (F_1), NOT (F_2)

(c) If F(x) is a WFF and x is free and is a domain variable then $(\exists x)$ (F(x)) and $\forall$(F(x)) are also WFF.

The definition of existential, universal quantifier and free bound variable are same as in tuple calculus.

Consider the following relations

Student (**RollNo**, **Name**, **Age**, **Address**, **Course**, **Year**, **Gender**, **DeptNo**)
q r s t u v w x

Department (**DeptNo**, **DName**)
n i

Example 1: List the Age and Address of student whose course is CSE and Name is Rajesh

{st | (∃u) (∃r) (Student) (qrstuvwx) AND u = 'CSE' AND r = 'Rajesh')}

In above query (RollNo. = q, Name = r, Age = s, Address = t, Course = u, Year = v Grade = w, DeptNo = x)

Example 2: Retrieve Roll No., Name of girl student in CSE department.

{qr | (∃w)(∃i)(∃x)(∃n) Student (qrstuvwx) AND Department (ni) AND w = 'F' AND i = 'CSE' AND x = n}

4.7.3 Relational Algebra Vs. Relational Calculus

(MDU May 2012; UPTU 2011-12)

Codd in 1971 claimed that relational calculus was superior to relational algebra. The advantages and differences of among them are described below: